General notes:

1. The MDP transition table is described by a custom language that follows the rules
    - A block of State -> Action rows (Ex.: Rested > Workout)
    - A separator row of "==========="
    - A block of Action to Reward and Environment state outcome rows (Ex.: Workout > 1 Tired)
2. The generated dot file has following conventions:
    - Environment outcomes are labeled with the reward and are colored brown
    - Actions are black arrows labeled with the action name
    - States are represented by labeled circles

Workout

The idea is to make a workout RL training scheduler that will prompt the user to train as much as possible without making the user declare "too tired to work anymore". Is a simplified idea that ignores the injuries and reluctant users.

To properly work the following limitations are in place:

- the "Resting" should be fixed time intervals.
- the "Workout" sessions should be similar routines.
- the time interval between "StartingNewDay" action and "EndOfDay" state should be identical between days.

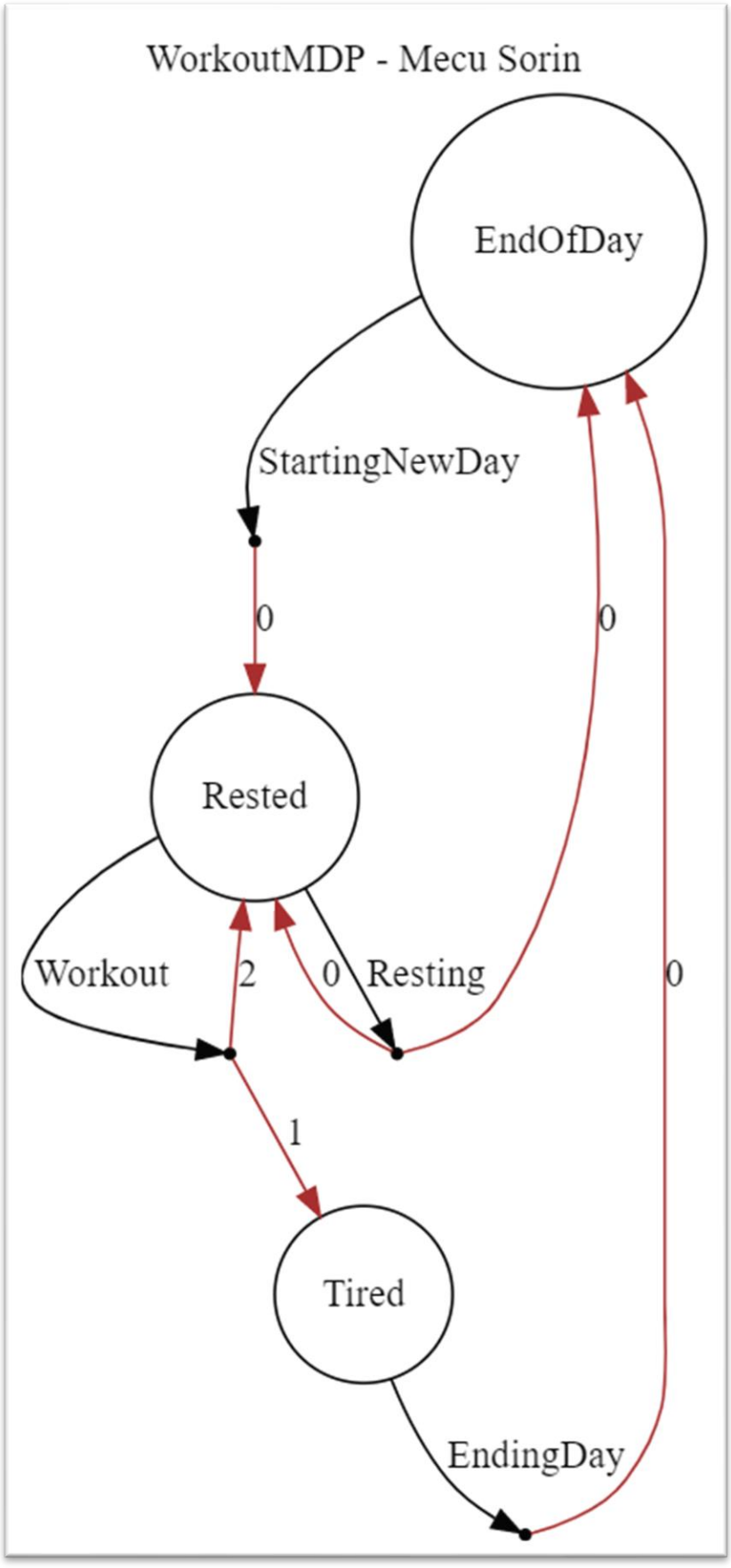
The transition table:

```
Rested > Workout
Rested > Resting
Tired > EndingDay
EndOfDay > StartingNewDay
============================================================
Workout > 2 Rested
Workout > 1 Tired
Resting > 0 Rested
Resting > 0 EndOfDay
EndingDay > 0 EndOfDay
StartingNewDay > 0 Rested
```

Generated dot file:

```
digraph WorkoutMDP {
   {
      node [ shape=circle ]
         EndOfDay
```

```
        Rested
        Tired

    node [shape=point]
        EndingDay
        Resting
        StartingNewDay
        Workout
  }

  Rested -> Workout [label="Workout"]
  Rested -> Resting [label="Resting"]
  Tired -> EndingDay [label="EndingDay"]
  EndOfDay -> StartingNewDay [label="StartingNewDay"]

  Workout -> Rested [label="2", color=brown]
  Workout -> Tired [label="1", color=brown]
  Resting -> Rested [label="0", color=brown]
  Resting -> EndOfDay [label="0", color=brown]
  EndingDay -> EndOfDay [label="0", color=brown]
  StartingNewDay -> Rested [label="0", color=brown]

  labelloc="t"
  label="WorkoutMDP - Mecu Sorin"
}
```


WorkoutMDP - Mecu Sorin
EndOfDay
StartingNewDay
0
Rested
Workout
2
0
Resting
0
0
1
Tired
EndingDay

Barrel Organ

The idea is to make a RL echoing device. Assuming a sound output equipment that is limited in capabilities, we can build a device that approximates the sound input. The flow will be to provide the required frequencies levels to be played as a state (sampling from a song for example) and the RL agent needs to use one of the actions available (for each available channel of the output equipment one of 2 actions **Play** or **Silence**). The "environment" will record the sound generated and compute the square sum on all the frequencies of difference between the input and the generated output, and feed it back with negative sign as a reward to the agent along with the next sample(new state – a list of all frequencies levels that the agent need to reproduce). The purpose will be to maximize the rewards, meaning the delta between the output sound and the input sound to be minimal.

The output device can be a combination of

- actuators on piano keys, xylophone keys, etc.
- valves mounted on blowing musical instruments, etc.

A state consists of an array of levels for all monitored frequencies.

An action consists of an array of all channels available from the output device, with a state that implies either "Play" or "Ignore".

A simplified version of the idea is to have a single frequency for both input and output, with the mention that the output device is an actuated bell. The sound recorder has just 2 levels: high and low, so the delta can be either 1 or 0.

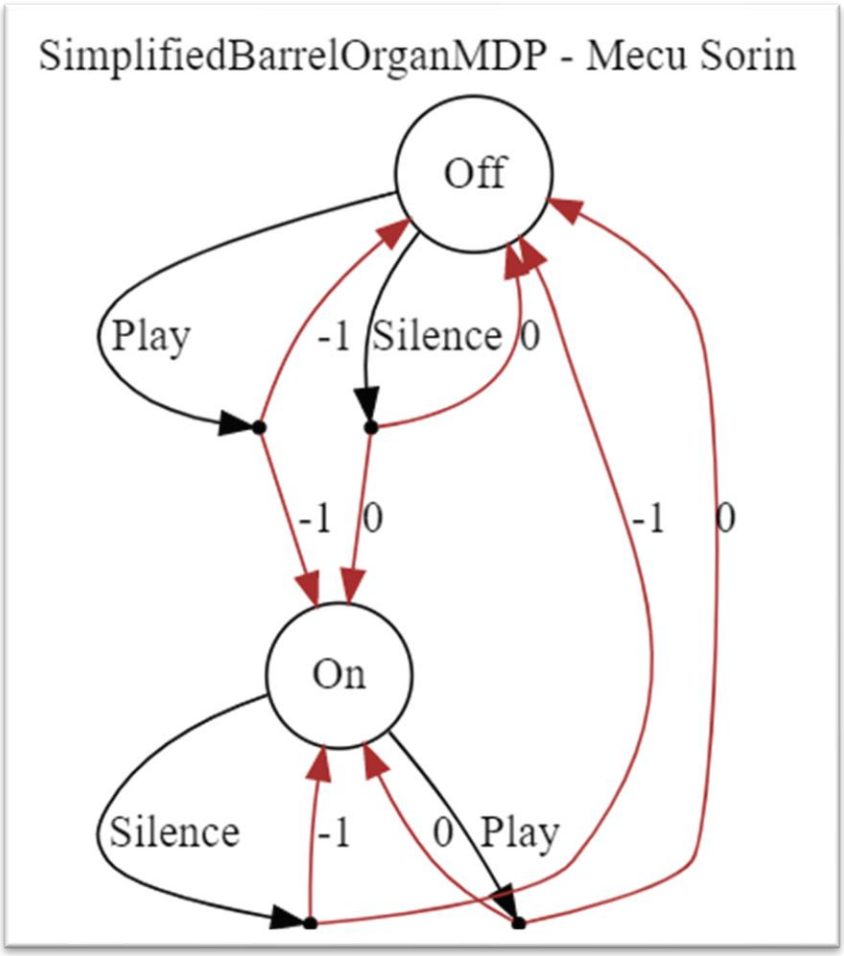
The transition table:

```
Off > Play_Off
Off > Silence_Off
On > Play_On
On > Silence_On
====================
Play_Off > -1 Off
Play_Off > -1 On
Silence_Off > 0 Off
Silence_Off > 0 On
Play_On > 0 Off
Play_On > 0 On
Silence_On > -1 Off
Silence_On > -1 On
```

Generated dot file:

```
digraph SimplifiedBarrelOrganMDP {
  {
    node [ shape=circle ]
```

```
        Off
        On

    node [shape=point]
        Play_Off
        Play_On
        Silence_Off
        Silence_On
  }

  Off -> Play_Off [label="Play"]
  Off -> Silence_Off [label="Silence"]
  On -> Play_On [label="Play"]
  On -> Silence_On [label="Silence"]

  Play_Off -> Off [label="-1", color=brown]
  Play_Off -> On [label="-1", color=brown]
  Silence_Off -> Off [label="0", color=brown]
  Silence_Off -> On [label="0", color=brown]
  Play_On -> Off [label="0", color=brown]
  Play_On -> On [label="0", color=brown]
  Silence_On -> Off [label="-1", color=brown]
  Silence_On -> On [label="-1", color=brown]

  labelloc="t"
  label="SimplifiedBarrelOrganMDP - Mecu Sorin"
}
```



Mosquitoes obliterator 😊

The idea is to make a RL agent that decides when to vacuum a mosquito. The device is composed by a vacuum cleaner, a ventilator, a video camera and an adapter that knows how to start and stop the vacuum cleaner and the ventilator. The vacuum cleaner nose is pointing to the ventilator at some distance, the camera is mounted on the ventilator side pointing towards the vacuum cleaner nose (the single reason for change in image captures to be mosquitoes flying). The vacuum cleaner and the ventilator will be controlled in the same time meaning that either both are working or both are off

States will be composed by a matrix with the offset between the previous frame and the current frame, the state of the devices (on|off)

Actions will be one of **DoNothing** | **StartDevices** | **StopDevices** limited by the current state (meaning if the devices are **ON** the **StartDevices** will not be available, and if devices are **OFF** the **StopDevices** will not be an available action)

The rewards will be 100 if the devices are on and the offset matrix is having a shape exactly on top of the vacuum cleaner nose(until a better way to count trapped mosquitoes), otherwise -1 if the devices are on (to not allow the agent to cheat by keeping the devices on all the time).

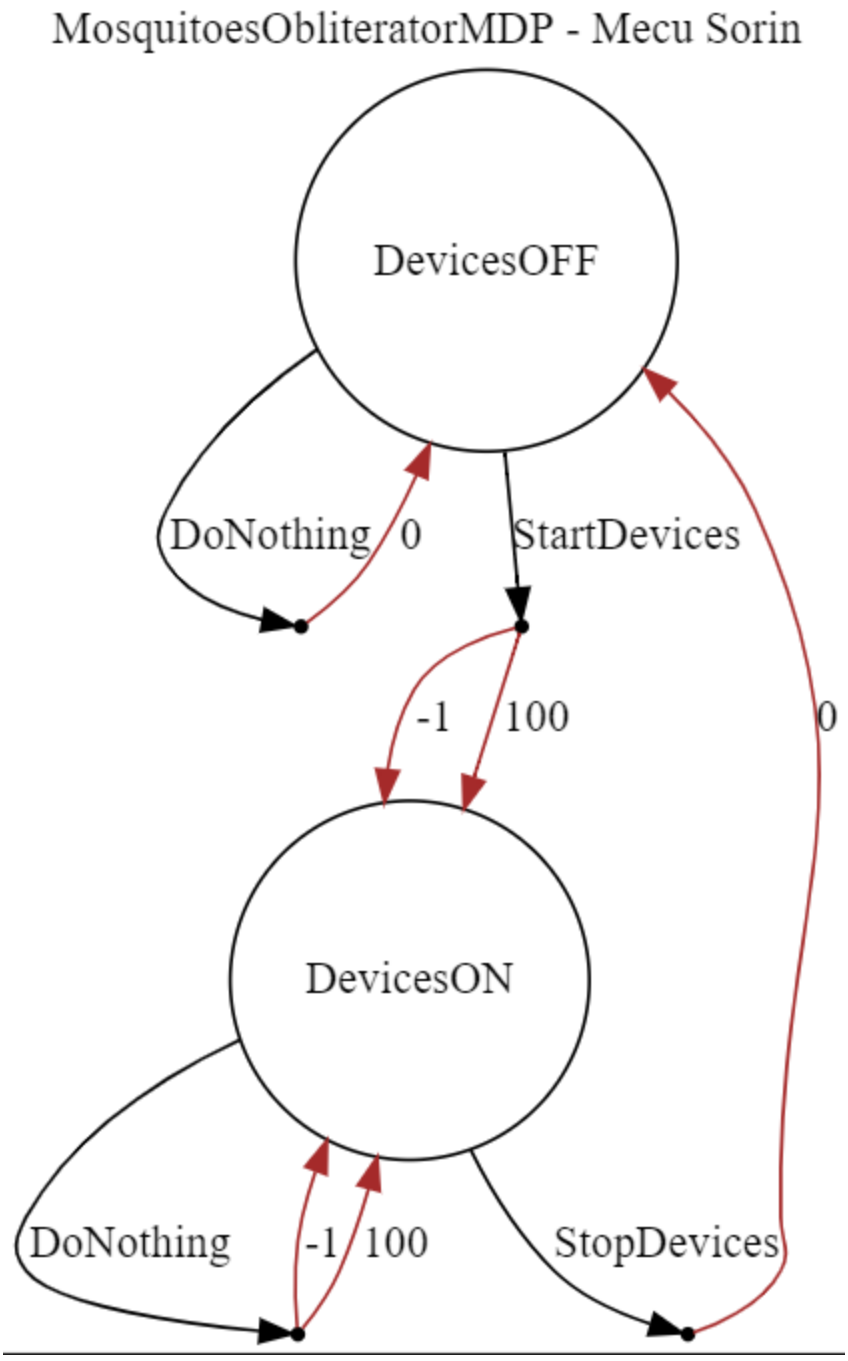
The transition table:

```
DevicesON > DoNothing_ON
DevicesON > StopDevices
DevicesOFF > DoNothing_OFF
DevicesOFF > StartDevices
=======================
DoNothing_ON > -1 DevicesON
DoNothing_ON > 100 DevicesON
DoNothing_OFF > 0 DevicesOFF
StartDevices > -1 DevicesON
StartDevices > 100 DevicesON
StopDevices > 0 DevicesOFF
```

Generated dot file:

```
digraph MosquitoesObliteratorMDP {
   {
      node [ shape=circle ]
         DevicesOFF
         DevicesON

      node [shape=point]
         DoNothing_OFF
         DoNothing_ON
         StartDevices
         StopDevices
   }

   DevicesON -> DoNothing_ON [label="DoNothing"]
   DevicesON -> StopDevices [label="StopDevices"]
   DevicesOFF -> DoNothing_OFF [label="DoNothing"]
   DevicesOFF -> StartDevices [label="StartDevices"]

   DoNothing_ON -> DevicesON [label="-1", color=brown]
   DoNothing_ON -> DevicesON [label="100", color=brown]
   DoNothing_OFF -> DevicesOFF [label="0", color=brown]
   StartDevices -> DevicesON [label="-1", color=brown]
   StartDevices -> DevicesON [label="100", color=brown]
   StopDevices -> DevicesOFF [label="0", color=brown]

   labelloc="t"
```

```
    label="MosquitoesObliteratorMDP - Mecu Sorin"
}
```



Don’t worry, EXTREME PREJUDICE happened to all mosquitoes involved in the experiment - in my mind.